

Sequential Neural Networks as Automata

William Merrill*

Yale University, New Haven, CT, USA

Allen Institute for Artificial Intelligence, Seattle, WA, USA

`william.merrill@yale.edu`

Abstract

This work attempts to explain the types of computation that neural networks can perform by relating them to automata. We first define what it means for a real-time network with bounded precision to accept a language. A measure of network memory follows from this definition. We then characterize the classes of languages acceptable by various recurrent networks, attention, and convolutional networks. We find that LSTMs function like counter machines and relate convolutional networks to the subregular hierarchy. Overall, this work attempts to increase our understanding and ability to interpret neural networks through the lens of theory. These theoretical insights help explain neural computation, as well as the relationship between neural networks and natural language grammar.

1 Introduction

In recent years, neural networks have achieved tremendous success on a variety of natural language processing (NLP) tasks. Neural networks employ continuous distributed representations of linguistic data, which contrast with classical discrete methods. While neural methods work well, one of the downsides of the distributed representations that they utilize is interpretability. It is hard to tell what kinds of computation a model is capable of, and when a model is working, it is hard to tell what it is doing.

This work aims to address such issues of interpretability by relating sequential neural networks to forms of computation that are more well understood. In theoretical computer science, the computational capacities of many different kinds of automata formalisms are clearly established. Moreover, the Chomsky hierarchy links natural

language to such automata-theoretic languages (Chomsky, 1956). Thus, relating neural networks to automata both yields insight into what general forms of computation such models can perform, as well as how such computation relates to natural language grammar.

Recent work has begun to investigate what kinds of automata-theoretic computations various types of neural networks can simulate. Weiss et al. (2018) propose a connection between long short-term memory networks (LSTMs) and counter automata. They provide a construction by which the LSTM can simulate a simplified variant of a counter automaton. They also demonstrate that LSTMs can learn to increment and decrement their cell state as counters in practice. Peng et al. (2018), on the other hand, describe a connection between the gating mechanisms of several recurrent neural network (RNN) architectures and weighted finite-state acceptors.

This paper follows Weiss et al. (2018) by analyzing the expressiveness of neural network acceptors under asymptotic conditions. We formalize asymptotic language acceptance, as well as an associated notion of network memory. We use this theory to derive computation upper bounds and automata-theoretic characterizations for several different kinds of recurrent neural networks (Section 3), as well as other architectural variants like attention (Section 4) and convolutional networks (CNNs) (Section 5). This leads to a fairly complete automata-theoretic characterization of sequential neural networks.

In Section 6, we report empirical results investigating how well these asymptotic predictions describe networks with continuous activations learned by gradient descent. In some cases, networks behave according to the theoretical predictions, but we also find cases where there is gap between the asymptotic characterization and ac-

*Work completed while the author was at Yale University.

tual network behavior.

Still, discretizing neural networks using an asymptotic analysis builds intuition about how the network computes. Thus, this work provides insight about the types of computations that sequential neural networks can perform through the lens of formal language theory. In so doing, we can also compare the notions of grammar expressible by neural networks to formal models that have been proposed for natural language grammar.

2 Introducing the Asymptotic Analysis

To investigate the capacities of different neural network architectures, we need to first define what it means for a neural network to accept a language. There are a variety of ways to formalize language acceptance, and changes to this definition lead to dramatically different characterizations.

In their analysis of RNN expressiveness, Siegelmann and Sontag (1992) allow RNNs to perform an unbounded number of recurrent steps even after the input has been consumed. Furthermore, they assume that the hidden units of the network can have arbitrarily fine-grained precision. Under this very general definition of language acceptance, Siegelmann and Sontag (1992) found that even a simple recurrent network (SRN) can simulate a Turing machine.

We want to impose the following constraints on neural network computation, which are more realistic to how networks are trained in practice (Weiss et al., 2018):

1. *Real-time*: The network performs one iteration of computation per input symbol.
2. *Bounded precision*: The value of each cell in the network is representable by $O(\log n)$ bits on sequences of length n .

Informally, a *neural sequence acceptor* is a network which reads a variable-length sequence of characters and returns the probability that the input sequence is a valid sentence in some formal language. More precisely, we can write:

Definition 2.1 (Neural sequence acceptor). Let \mathbf{X} be a matrix representation of a sentence where each row is a one-hot vector over an alphabet Σ . A neural sequence acceptor $\hat{\mathbb{I}}$ is a family of functions parameterized by weights θ . For each θ and \mathbf{X} , the function $\hat{\mathbb{I}}^\theta$ takes the form

$$\hat{\mathbb{I}}^\theta : \mathbf{X} \mapsto p \in (0, 1).$$

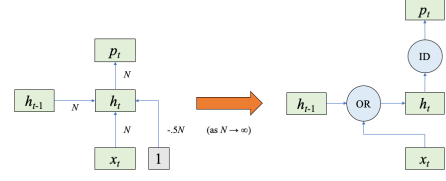


Figure 1: With sigmoid activations, the network on the left accepts a sequence of bits if and only if $x_t = 1$ for some t . On the right is the discrete computation graph that the network approaches asymptotically.

In this definition, $\hat{\mathbb{I}}$ corresponds to a general architecture like an LSTM, whereas $\hat{\mathbb{I}}^\theta$ represents a specific network, such as an LSTM with weights that have been learned from data.

In order to get an acceptance decision from this kind of network, we will consider what happens as the magnitude of its parameters gets very large. Under these asymptotic conditions, the internal connections of the network approach a discrete computation graph, and the probabilistic output approaches the indicator function of some language (Figure 1).

Definition 2.2 (Asymptotic acceptance). Let L be a language with indicator function $\mathbb{1}_L$. A neural sequence acceptor $\hat{\mathbb{I}}$ with weights θ asymptotically accepts L if

$$\lim_{N \rightarrow \infty} \hat{\mathbb{I}}^{N\theta} = \mathbb{1}_L.$$

Note that the limit of $\hat{\mathbb{I}}^{N\theta}$ represents the function that $\hat{\mathbb{I}}^{N\theta}$ converges to pointwise.¹

Discretizing the network in this way lets us analyze it as an automaton. We can also view this discretization as a way of bounding the precision that each unit in the network can encode, since it is forced to act as a discrete unit instead of a continuous value. This prevents complex fractal representations that rely on infinite precision. We will see later that, for every architecture considered, this definition ensures that the value of every unit in the network is representable in $O(\log n)$ bits on sequences of length n .

It is important to note that real neural networks can learn strategies not allowed by the asymptotic definition. Thus, this way of analyzing neural networks is not completely faithful to their practical

¹https://en.wikipedia.org/wiki/Pointwise_convergence

usage. In Section 6, we discuss empirical studies investigating how trained networks compare to the asymptotic predictions. While we find evidence of networks learning behavior that is not asymptotically stable, adding noise to the network during training seems to make it more difficult for the network to learn non-asymptotic strategies.

Consider a neural network that asymptotically accepts some language. For any given length, we can pick weights for the network such that it will correctly decide strings shorter than that length (Theorem A.1).

Analyzing a network’s asymptotic behavior also gives us a notion of the network’s memory. Weiss et al. (2018) illustrate how the LSTM’s additive cell update gives it more effective memory than the squashed state of an SRN or GRU for solving counting tasks. We generalize this concept of memory capacity as *state complexity*. Informally, the state complexity of a node within a network represents the number of values that the node can achieve asymptotically as a function of the sequence length n . For example, the LSTM cell state will have $O(n^k)$ state complexity (Theorem 3.3), whereas the state of other recurrent networks has $O(1)$ (Theorem 3.1).

State complexity applies to a *hidden state* sequence, which we can define as follows:

Definition 2.3 (Hidden state). For any sentence \mathbf{X} , let n be the length of \mathbf{X} . For $1 \leq t \leq n$, the k -length hidden state \mathbf{h}_t with respect to parameters θ is a sequence of functions given by

$$\mathbf{h}_t^\theta : \mathbf{X} \mapsto \mathbf{v}_t \in \mathbb{R}^k.$$

Often, a sequence acceptor can be written as a function of an intermediate hidden state. For example, the output of the recurrent layer acts as a hidden state in an LSTM language acceptor. In recurrent architectures, the value of the hidden state is a function of the preceding prefix of characters, but with convolution or attention, it can depend on characters occurring after index t .

The *state complexity* is defined as the cardinality of the *configuration set* of such a hidden state:

Definition 2.4 (Configuration set). For all n , the configuration set of hidden state \mathbf{h}_n with respect to parameters θ is given by

$$M(\mathbf{h}_n^\theta) = \left\{ \lim_{N \rightarrow \infty} \mathbf{h}_n^{N\theta}(\mathbf{X}) \mid n = |\mathbf{X}| \right\}.$$

where $|\mathbf{X}|$ is the length, or height, of the sentence matrix \mathbf{X} .

Definition 2.5 (Fixed state complexity). For all n , the fixed state complexity of hidden state \mathbf{h}_n with respect to parameters θ is given by

$$\mathfrak{M}(\mathbf{h}_n^\theta) = |M(\mathbf{h}_n^\theta)|.$$

Definition 2.6 (General state complexity). For all n , the general state complexity of hidden state \mathbf{h}_n is given by

$$\mathfrak{M}(\mathbf{h}_n) = \max_{\theta} \mathfrak{M}(\mathbf{h}_n^\theta).$$

To illustrate these definitions, consider a simplified recurrent mechanism based on the LSTM cell. The architecture is parameterized by a vector $\theta \in \mathbb{R}^2$. At each time step, the network reads a bit x_t and computes

$$f_t = \sigma(\theta_1 x_t) \quad (1)$$

$$i_t = \sigma(\theta_2 x_t) \quad (2)$$

$$h_t = f_t h_{t-1} + i_t. \quad (3)$$

When we set $\theta^+ = \langle 1, 1 \rangle$, h_t asymptotically computes the sum of the preceding inputs. Because this sum can evaluate to any integer between 0 and n , $h_n^{\theta^+}$ has a fixed state complexity of

$$\mathfrak{M}(h_n^{\theta^+}) = O(n). \quad (4)$$

However, when we use parameters $\theta^{\text{Id}} = \langle -1, 1 \rangle$, we get a reduced network where $h_t = x_t$ asymptotically. Thus,

$$\mathfrak{M}(h_n^{\theta^{\text{Id}}}) = O(1). \quad (5)$$

Finally, the general state complexity is the maximum fixed complexity, which is $O(n)$.

For any neural network hidden state, the state complexity is at most $2^{O(n)}$ (Theorem A.2). This means that the value of the hidden unit can be encoded in $O(n)$ bits. Moreover, for every specific architecture considered, we observe that each fixed-length state vector has at most $O(n^k)$ state complexity, or, equivalently, can be represented in $O(\log n)$ bits.

Architectures that have exponential state complexity, such as the transformer, do so by using a variable-length hidden state. State complexity generalizes naturally to a variable-length hidden state, with the only difference being that \mathbf{h}_t (Definition 2.3) becomes a sequence of variably sized objects rather than a sequence of fixed-length vectors.

Now, we consider what classes of languages different neural networks can accept asymptotically. We also analyze different architectures in terms of state complexity. The theory that emerges from these tools enables better understanding of the computational processes underlying neural sequence models.

3 Recurrent Neural Networks

As previously mentioned, RNNs are Turing-complete under an unconstrained definition of acceptance (Siegelmann and Sontag, 1992). The classical reduction of a Turing machine to an RNN relies on two unrealistic assumptions about RNN computation (Weiss et al., 2018). First, the number of recurrent computations must be unbounded in the length of the input, whereas, in practice, RNNs are almost always trained in a real-time fashion. Second, it relies heavily on infinite precision of the network’s logits. We will see that the asymptotic analysis, which restricts computation to be real-time and have bounded precision, severely narrows the class of formal languages that an RNN can accept.

3.1 Simple Recurrent Networks

The SRN, or Elman network, is the simplest type of RNN (Elman, 1990):

Definition 3.1 (SRN layer).

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}). \quad (6)$$

A well-known problem with SRNs is that they struggle with long-distance dependencies. One explanation of this is the vanishing gradient problem, which motivated the development of more sophisticated architectures like the LSTM (Hochreiter and Schmidhuber, 1997). Another shortcoming of the SRN is that, in some sense, it has less memory than the LSTM. This is because, while both architectures have a fixed number of hidden units, the SRN units remain between -1 and 1 , whereas the value of each LSTM cell can grow unboundedly (Weiss et al., 2018). We can formalize this intuition by showing that the SRN has finite state complexity:

Theorem 3.1 (SRN state complexity). *For any length n , the SRN cell state $\mathbf{h}_n \in \mathbb{R}^k$ has state complexity*

$$\mathfrak{N}(\mathbf{h}_n) \leq 2^k = O(1).$$

Proof. For every n , each unit of \mathbf{h}_n will be the output of a \tanh . In the limit, it can achieve either -1 or 1 . Thus, for the full vector, the number of configurations is bounded by 2^k . \square

It also follows from Theorem 3.1 that the languages asymptotically acceptable by an SRN are a subset of the finite-state (i.e. regular) languages. Lemma B.1 provides the other direction of this containment. Thus, SRNs are equivalent to finite-state automata.

Theorem 3.2 (SRN characterization). *Let $L(\text{SRN})$ denote the languages acceptable by an SRN, and RL the regular languages. Then,*

$$L(\text{SRN}) = \text{RL}.$$

This characterization is quite diminished compared to Turing completeness. It is also more descriptive of what SRNs can express in practice. We will see that LSTMs, on the other hand, are strictly more powerful than the regular languages.

3.2 Long Short-Term Memory Networks

An LSTM is a recurrent network with a complex gating mechanism that determines how information from one time step is passed to the next. Originally, this gating mechanism was designed to remedy the vanishing gradient problem in SRNs, or, equivalently, to make it easier for the network to remember long-term dependencies (Hochreiter and Schmidhuber, 1997). Due to strong empirical performance on many language tasks, LSTMs have become a canonical model for NLP.

Weiss et al. (2018) suggest that another advantage of the LSTM architecture is that it can use its cell state as counter memory. They point out that this constitutes a real difference between the LSTM and the GRU, whose update equations do not allow it to increment or decrement its memory units. We will further investigate this connection between LSTMs and counter machines.

Definition 3.2 (LSTM layer).

$$\mathbf{f}_t = \sigma(\mathbf{W}^f \mathbf{x}_t + \mathbf{U}^f \mathbf{h}_{t-1} + \mathbf{b}^f) \quad (7)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^i \mathbf{x}_t + \mathbf{U}^i \mathbf{h}_{t-1} + \mathbf{b}^i) \quad (8)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o \mathbf{x}_t + \mathbf{U}^o \mathbf{h}_{t-1} + \mathbf{b}^o) \quad (9)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}^c \mathbf{x}_t + \mathbf{U}^c \mathbf{h}_{t-1} + \mathbf{b}^c) \quad (10)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (11)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{c}_t). \quad (12)$$

In (12), we set f to either the identity or \tanh (Weiss et al., 2018), although \tanh is more standard in practice. The vector \mathbf{h}_t is the output that is received by the next layer, and \mathbf{c}_t is an unexposed memory vector called the cell state.

Theorem 3.3 (LSTM state complexity). *The LSTM cell state $\mathbf{c}_n \in \mathbb{R}^k$ has state complexity*

$$\mathfrak{M}(\mathbf{c}_n) = O(n^k).$$

Proof. At each time step t , we know that the configuration sets of \mathbf{f}_t , \mathbf{i}_t , and \mathbf{o}_t are each subsets of $\{0, 1\}^k$. Similarly, the configuration set of $\tilde{\mathbf{c}}_t$ is a subset of $\{-1, 1\}^k$. This allows us to rewrite the elementwise recurrent update as

$$\lim_{N \rightarrow \infty} [\mathbf{c}_t]_i = \lim_{N \rightarrow \infty} [\mathbf{f}_t]_i [\mathbf{c}_{t-1}]_i + [\mathbf{i}_t]_i [\tilde{\mathbf{c}}_t]_i \quad (13)$$

$$= \lim_{N \rightarrow \infty} a[\mathbf{c}_{t-1}]_i + b \quad (14)$$

where $a \in \{0, 1\}$ and $b \in \{-1, 0, 1\}$.

Let S_t be the configuration set of $[\mathbf{c}_t]_i$. At each time step, we have exactly two ways to produce a new value in S_t that was not in S_{t-1} : either we decrement the minimum value in S_{t-1} or increment the maximum value. It follows that

$$|S_t| = 2 + |S_{t-1}| \quad (15)$$

$$\implies |S_n| = O(n). \quad (16)$$

For all k units of the cell state, we get

$$\mathfrak{M}(\mathbf{c}_n) \leq |S_n|^k = O(n^k). \quad (17)$$

□

The construction in Theorem 3.3 produces a counter machine whose counter and state update functions are linearly separable. Thus, we have an upper bound on the expressive power of the LSTM:

Theorem 3.4 (LSTM upper bound). *Let CL be the real-time counter languages (Fischer, 1966; Fischer et al., 1968). Then,*

$$L(\text{LSTM}) \subseteq \text{CL}.$$

Theorem 3.4 constitutes a very tight upper bound on the expressiveness of LSTM computation. Asymptotically, LSTMs are not powerful enough to model even the deterministic context-free language $w\#w^R$.

Weiss et al. (2018) show how the LSTM can simulate a simplified variant of the counter machine. Combining these results, we see that

the asymptotic expressiveness of the LSTM falls somewhere between the general and simplified counter languages. This suggests counting is a good way to understand the behavior of LSTMs.

3.3 Gated Recurrent Units

The GRU is a popular gated recurrent architecture that is in many ways similar to the LSTM (Cho et al., 2014). Rather than having separate forget and input gates, the GRU utilizes a single gate that controls both functions.

Definition 3.3 (GRU layer).

$$\mathbf{z}_t = \sigma(\mathbf{W}^z \mathbf{x}_t + \mathbf{U}^z \mathbf{h}_{t-1} + \mathbf{b}^z) \quad (18)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}^r \mathbf{x}_t + \mathbf{U}^r \mathbf{h}_{t-1} + \mathbf{b}^r) \quad (19)$$

$$\mathbf{u}_t = \tanh(\mathbf{W}^u \mathbf{x}_t + \mathbf{U}^u (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}^u) \quad (20)$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{u}_t. \quad (21)$$

Weiss et al. (2018) observe that GRUs do not exhibit the same counter behavior as LSTMs on languages like $a^n b^n$. As with the SRN, the GRU state is squashed between -1 and 1 (20). Taken together, Lemmas C.1 and C.2 show that GRUs, like SRNs, are finite-state.

Theorem 3.5 (GRU characterization).

$$L(\text{GRU}) = \text{RL}.$$

3.4 RNN Complexity Hierarchy

Synthesizing all of these results, we get the following complexity hierarchy:

$$\text{RL} = L(\text{SRN}) = L(\text{GRU}) \quad (22)$$

$$\subset \text{SCL} \subseteq L(\text{LSTM}) \subseteq \text{CL}. \quad (23)$$

Basic recurrent architectures have finite state, whereas the LSTM is strictly more powerful than a finite-state machine.

4 Attention

Attention is a popular enhancement to sequence-to-sequence (seq2seq) neural networks (Bahdanau et al., 2014; Chorowski et al., 2015; Luong et al., 2015). Attention allows a network to recall specific encoder states while trying to produce output. In the context of machine translation, this mechanism models the alignment between words in the source and target languages. More recent work has found that “attention is all you need” (Vaswani et al., 2017; Radford et al., 2018). In other words,

networks with only attention and no recurrent connections perform at the state of the art on many tasks.

An attention function maps a query vector and a sequence of paired key-value vectors to a weighted combination of the values. This lookup function is meant to retrieve the values whose keys resemble the query.

Definition 4.1 (Dot-product attention). For any n , define a query vector $\mathbf{q} \in \mathbb{R}^l$, matrix of key vectors $\mathbf{K} \in \mathbb{R}^{nl}$, and matrix of value vectors $\mathbf{V} \in \mathbb{R}^{nk}$. Dot-product attention is given by

$$\text{attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{q}\mathbf{K}^T)\mathbf{V}.$$

In Definition 4.1, softmax creates a vector of similarity scores between the query \mathbf{q} and the key vectors in \mathbf{K} . The output of attention is thus a weighted sum of the value vectors where the weight for each value represents its relevance.

In practice, the dot product $\mathbf{q}\mathbf{K}^T$ is often scaled by the square root of the length of the query vector (Vaswani et al., 2017). However, this is only done to improve optimization and has no effect on expressiveness. Therefore, we consider the unscaled version.

In the asymptotic case, attention reduces to a weighted average of the values whose keys maximally resemble the query. This can be viewed as an $\arg \max$ operation.

Theorem 4.1 (Asymptotic attention). Let t_1, \dots, t_m be the subsequence of time steps that maximize $\mathbf{q}\mathbf{k}_t$.² Asymptotically, attention computes

$$\lim_{N \rightarrow \infty} \text{attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \lim_{N \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m \mathbf{v}_{t_i}.$$

Corollary 4.1.1 (Asymptotic attention with unique maximum). If $\mathbf{q}\mathbf{k}_t$ has a unique maximum over $1 \leq t \leq n$, then attention asymptotically computes

$$\lim_{N \rightarrow \infty} \text{attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \lim_{N \rightarrow \infty} \arg \max_{\mathbf{v}_t} \mathbf{q}\mathbf{k}_t.$$

Now, we analyze the effect of adding attention to an acceptor network. Because we are concerned with language acceptance instead of transduction, we consider a simplified seq2seq attention model where the output sequence has length 1:

²To be precise, we can define a maximum over the similarity scores according to the order given by

$$f > g \iff \lim_{N \rightarrow \infty} f(N) - g(N) > 0. \quad (24)$$

Definition 4.2 (Attention layer). Let the hidden state $\mathbf{v}_1, \dots, \mathbf{v}_n$ be the output of an encoder network where the union of the asymptotic configuration sets over all \mathbf{v}_t is finite. We attend over \mathbf{V}_t , the matrix stacking $\mathbf{v}_1, \dots, \mathbf{v}_t$, by computing

$$\mathbf{h}_t = \text{attn}(\mathbf{W}^q \mathbf{v}_t, \mathbf{V}_t, \mathbf{V}_t).$$

In this model, \mathbf{h}_t represents a summary of the relevant information in the prefix $\mathbf{v}_1, \dots, \mathbf{v}_t$. The query that is used to attend at time t is a simple linear transformation of \mathbf{v}_t .

In addition to modeling alignment, attention improves a bounded-state model by providing additional memory. By converting the state of the network to a growing sequence \mathbf{V}_t instead of a fixed length vector \mathbf{v}_t , attention enables $2^{\Theta(n)}$ state complexity.

Theorem 4.2 (Encoder state complexity). The full state of the attention layer has state complexity

$$\mathfrak{M}(\mathbf{V}_n) = 2^{\Theta(n)}.$$

The $O(n^k)$ complexity of the LSTM architecture means that it is impossible for LSTMs to copy or reverse long strings. The exponential state complexity provided by attention enables copying, which we can view as a simplified version of machine translation. Thus, it makes sense that attention is almost universal in machine translation architectures. The additional memory introduced by attention might also allow more complex hierarchical representations.

A natural follow-up question to Theorem 4.2 is whether this additional complexity is preserved in the attention summary vector \mathbf{h}_n . Attending over \mathbf{V}_n does not preserve exponential state complexity. Instead, we get an $O(n^2)$ summary of \mathbf{V}_n .

Theorem 4.3 (Summary state complexity). The attention summary vector has state complexity

$$\mathfrak{M}(\mathbf{h}_n) = O(n^2).$$

With minimal additional assumptions, we can show a more restrictive bound: namely, that the complexity of the summary vector is finite. Appendix D discusses this in more detail.

5 Convolutional Networks

While CNNs were originally developed for image processing (Krizhevsky et al., 2012), they are also

used to encode sequences. One popular application of this is to build character-level representations of words (Kim et al., 2016). Another example is the capsule network architecture of Zhao et al. (2018), which uses a convolutional layer as an initial feature extractor over a sentence.

Definition 5.1 (CNN acceptor).

$$\mathbf{h}_t = \tanh(\mathbf{W}^h(\mathbf{x}_{t-k} \parallel \dots \parallel \mathbf{x}_{t+k}) + \mathbf{b}^h) \quad (25)$$

$$\mathbf{h}_+ = \text{maxpool}(\mathbf{H}) \quad (26)$$

$$p = \sigma(\mathbf{W}^a \mathbf{h}_+ + \mathbf{b}^a). \quad (27)$$

In this network, the k -convolutional layer (25) produces a vector-valued sequence of outputs. This sequence is then collapsed to a fixed length by taking the maximum value of each filter over all the time steps (26).

The CNN acceptor is much weaker than the LSTM. Since the vector \mathbf{h}_t has finite state, we see that $L(\text{CNN}) \subseteq \text{RL}$. Moreover, simple regular languages like a^*ba^* are beyond the CNN (Lemma E.1). Thus, the subset relation is strict.

Theorem 5.1 (CNN upper bound).

$$L(\text{CNN}) \subset \text{RL}.$$

So, to arrive at a characterization of CNNs, we should move to subregular languages. In particular, we consider the strictly local languages (Rogers and Pullum, 2011).

Theorem 5.2 (CNN lower bound). *Let SL be the strictly local languages. Then,*

$$\text{SL} \subseteq L(\text{CNN}).$$

Notably, strictly local formalisms have been proposed as a computational model for phonological grammar (Heinz et al., 2011). We might take this to explain why CNNs have been successful at modeling character-level information.

However, Heinz et al. (2011) suggest that a generalization to the tier-based strictly local languages is necessary to account for the full range of phonological phenomena. Tier-based strictly local grammars can target characters in a specific tier of the vocabulary (e.g. vowels) instead of applying to the full string. While a single convolutional layer cannot utilize tiers, it is conceivable that a more complex architecture with recurrent connections could.

6 Empirical Results

In this section, we compare our theoretical characterizations for asymptotic networks to the empirical performance of trained neural networks with continuous logits.³

6.1 Counting

The goal of this experiment is to evaluate which architectures have memory beyond finite state. We train a language model on $a^n b^n c$ with $5 \leq n \leq 1000$ and test it on longer strings ($2000 \leq n \leq 2200$). Predicting the c character correctly while maintaining good overall accuracy requires $O(n)$ states. The results reported in Table 1 demonstrate that all recurrent models, with only two hidden units, find a solution to this task that generalizes at least over this range of string lengths.

Weiss et al. (2018) report failures in attempts to train SRNs and GRUs to accept counter languages, unlike what we have found. We conjecture that this stems not from the requisite memory, but instead from the different objective function we used. Our language modeling training objective is a robust and transferable learning target (Radford et al., 2019), whereas sparse acceptance classification might be challenging to learn directly for long strings.

Weiss et al. (2018) also observe that LSTMs use their memory as counters in a straightforwardly interpretable manner, whereas SRNs and GRUs do not do so in any obvious way. Despite this, our results show that SRNs and GRUs are nonetheless able to implement generalizable counter memory while processing strings of significant length. Because the strategies learned by these architectures are not asymptotically stable, however, their schemes for encoding counting are less interpretable.

6.2 Counting with Noise

In order to abstract away from asymptotically unstable representations, our next experiment investigates how adding noise to an RNN’s activations impacts its ability to count. For the SRN and GRU, noise is added to \mathbf{h}_{t-1} before computing \mathbf{h}_t , and for the LSTM, noise is added to \mathbf{c}_{t-1} . In either case, the noise is sampled from the distribution $N(0, 0.1^2)$.

³<https://github.com/viking-sudo-rm/nl-automata>

		No Noise		Noise	
	\mathfrak{M}	Acc	Acc on c	Acc	Acc on c
SRN	$O(1)$	100.0	100.0	49.9	100.0
GRU	$O(1)$	99.9	100.0	53.9	100.0
LSTM	$O(n^k)$	99.9	100.0	99.9	100.0

Table 1: Generalization performance of language models trained on $a^n b^n c$. Each model has 2 hidden units.

	\mathfrak{M}	Val Acc	Gen Acc
LSTM	$O(n^k)$	94.0	51.6
LSTM-Attn	$2^{\Theta(n)}$	100.0	51.7
LSTM	$O(n^k)$	92.5	73.3
StackNN	$2^{\Theta(n)}$	100.0	100.0

Table 2: Max validation and generalization accuracies on string reversal over 10 trials. The top section shows our seq2seq LSTM with and without attention. The bottom reports the LSTM and StackNN results of Hao et al. (2018). Each LSTM has 10 hidden units.

The results reported in the right column of Table 1 show that the noisy SRN and GRU now fail to count, whereas the noisy LSTM remains successful. Thus, the asymptotic characterization of each architecture matches the capacity of a trained network when a small amount of noise is introduced.

From a practical perspective, training neural networks with Gaussian noise is one way of improving generalization by preventing overfitting (Bishop, 1995; Noh et al., 2017). From this point of view, asymptotic characterizations might be more descriptive of the generalization capacities of regularized neural networks of the sort necessary to learn the patterns in natural language data as opposed to the unregularized networks that are typically used to learn the patterns in carefully curated formal languages.

6.3 Reversing

Another important formal language task for assessing network memory is string reversal. Reversing requires remembering a $\Theta(n)$ prefix of characters, which implies $2^{\Theta(n)}$ state complexity.

We frame reversing as a seq2seq transduction task, and compare the performance of an LSTM encoder-decoder architecture to the same architecture augmented with attention. We also report the results of Hao et al. (2018) for a stack neural network (StackNN), another architecture with $2^{\Theta(n)}$ state complexity (Lemma F.1).

Following Hao et al. (2018), the models were

trained on 800 random binary strings with length $\sim N(10, 2)$ and evaluated on strings with length $\sim N(50, 5)$. As can be seen in Table 2, the LSTM with attention achieves 100.0% validation accuracy, but fails to generalize to longer strings. In contrast, Hao et al. (2018) report that a stack neural network can learn and generalize string reversal flawlessly. In both cases, it seems that having $2^{\Theta(n)}$ state complexity enables better performance on this memory-demanding task. However, our seq2seq LSTMs appear to be biased against finding a strategy that generalizes to longer strings.

7 Conclusion

We have introduced asymptotic acceptance as a new way to characterize neural networks as automata of different sorts. It provides a useful and generalizable tool for building intuition about how a network works, as well as for comparing the formal properties of different architectures. Further, by combining asymptotic characterizations with existing results in mathematical linguistics, we can better assess the suitability of different architectures for the representation of natural language grammar.

We observe empirically, however, that this discrete analysis fails to fully characterize the range of behaviors expressible by neural networks. In particular, RNNs predicted to be finite-state solve a task that requires more than finite memory. On the other hand, introducing a small amount of noise into a network’s activations seems to prevent it from implementing non-asymptotic strategies. Thus, asymptotic characterizations might be a good model for the types of generalizable strategies that noise-regularized neural networks trained on natural language data can learn.

Acknowledgements

Thank you to Dana Angluin and Robert Frank for their insightful advice and support on this project.

References

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. [Neural machine translation by jointly learning to align and translate](#). *arXiv preprint arXiv:1409.0473*.
- Chris M. Bishop. 1995. [Training with noise is equivalent to Tikhonov regularization](#). *Neural Comput.*, 7(1):108–116.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. [Learning phrase representations using RNN encoder–decoder for statistical machine translation](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Noam Chomsky. 1956. [Three models for the description of language](#). *IRE Transactions on information theory*, 2(3):113–124.
- Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. [Attention-based models for speech recognition](#). In *Advances in neural information processing systems*, pages 577–585.
- Jeffrey L Elman. 1990. [Finding structure in time](#). *Cognitive science*, 14(2):179–211.
- Patrick C Fischer. 1966. [Turing machines with restricted memory access](#). *Information and Control*, 9(4):364–379.
- Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. 1968. [Counter machines and counter languages](#). *Mathematical systems theory*, 2(3):265–283.
- Yiding Hao, William Merrill, Dana Angluin, Robert Frank, Noah Amsel, Andrew Benz, and Simon Mendelsohn. 2018. [Context-free transductions with neural stacks](#). *arXiv preprint arXiv:1809.02836*.
- Jeffrey Heinz, Chetan Rawal, and Herbert G Tanner. 2011. [Tier-based strictly local constraints for phonology](#). In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 58–64. Association for Computational Linguistics.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. [Long short-term memory](#). *Neural computation*, 9(8):1735–1780.
- Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. [Character-aware neural language models](#). In *AAAI*, pages 2741–2749.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. [Imagenet classification with deep convolutional neural networks](#). In *Advances in neural information processing systems*, pages 1097–1105.
- Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. [Effective approaches to attention-based neural machine translation](#). *arXiv preprint arXiv:1508.04025*.
- Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. 2017. [Regularizing deep neural networks by noise: Its interpretation and optimization](#). In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5115–5124.
- Hao Peng, Roy Schwartz, Sam Thomson, and Noah A Smith. 2018. [Rational recurrences](#). *arXiv preprint arXiv:1808.09357*.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. [Improving language understanding by generative pre-training](#).
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. [Language models are unsupervised multitask learners](#). URL <https://openai.com/blog/better-language-models>.
- James Rogers and Geoffrey K Pullum. 2011. [Aural pattern recognition experiments and the subregular hierarchy](#). *Journal of Logic, Language and Information*, 20(3):329–342.
- Hava T. Siegelmann and Eduardo D. Sontag. 1992. [On the computational power of neural nets](#). In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pages 440–449, New York, NY, USA. ACM.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In *Advances in Neural Information Processing Systems*, pages 5998–6008.
- Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. [On the practical computational power of finite precision RNNs for language recognition](#). *CoRR*, abs/1805.04908.
- Wei Zhao, Jianbo Ye, Min Yang, Zeyang Lei, Suofei Zhang, and Zhou Zhao. 2018. [Investigating capsule networks with dynamic routing for text classification](#). *arXiv preprint arXiv:1804.00538*.

A Asymptotic Acceptance and State Complexity

Theorem A.1 (Arbitrary approximation). *Let $\hat{\mathbb{1}}$ be a neural sequence acceptor for L . For all m , there exist parameters θ_m such that, for any string $\mathbf{x}_1, \dots, \mathbf{x}_n$ with $n < m$,*

$$\left[\hat{\mathbb{1}}^{\theta_m}(\mathbf{X}) \right] = \mathbb{1}_L(\mathbf{X})$$

where $\lceil \cdot \rceil$ rounds to the nearest integer.

Proof. Consider a string \mathbf{X} . By the definition of asymptotic acceptance, there exists some number $M_{\mathbf{X}}$ which is the smallest number such that, for all $N \geq M_{\mathbf{X}}$,

$$\left| \hat{\mathbb{1}}^{N\theta}(\mathbf{X}) - \mathbb{1}_L(\mathbf{X}) \right| < \frac{1}{2} \quad (28)$$

$$\implies \left[\hat{\mathbb{1}}^{N\theta}(\mathbf{X}) \right] = \mathbb{1}_L(\mathbf{X}). \quad (29)$$

Now, let X_m be the set of sentences \mathbf{X} with length less than m . Since X_m is finite, we pick θ_m just by taking

$$\theta_m = \max_{\mathbf{X} \in X_m} M_{\mathbf{X}} \theta. \quad (30)$$

□

Theorem A.2 (General bound on state complexity). *Let \mathbf{h}_t be a neural network hidden state. For any length n , it holds that*

$$\mathfrak{M}(\mathbf{h}_n) = 2^{O(n)}.$$

Proof. The number of configurations of \mathbf{h}_n cannot be more than the number of distinct inputs to the network. By construction, each \mathbf{x}_t is a one-hot vector over the alphabet Σ . Thus, the state complexity is bounded according to

$$\mathfrak{M}(\mathbf{h}_n) \leq |\Sigma|^n = 2^{O(n)}.$$

□

B SRN Lemmas

Lemma B.1 (SRN lower bound).

$$\text{RL} \subseteq L(\text{SRN}).$$

Proof. We must show that any language acceptable by a finite-state machine is SRN-acceptable. We need to asymptotically compute a representation of the machine's state in \mathbf{h}_t . We do this by storing all values of the following finite predicate at each time step:

$$\tilde{\partial}_t(i, \alpha) \iff q_{t-1}(i) \wedge x_t = \alpha \quad (31)$$

where $q_t(i)$ is true if the machine is in state i at time t .

Let F be the set of accepting states for the machine, and let δ^{-1} be the inverse transition relation. Assuming \mathbf{h}_t asymptotically computes $\tilde{\partial}_t$, we can decide to accept or reject in the final layer according to the linearly separable disjunction

$$a_t \iff \bigvee_{i \in F} \bigvee_{\langle j, \alpha \rangle \in \delta^{-1}(i)} \tilde{\partial}_t(j, \alpha). \quad (32)$$

We now show how to recurrently compute $\tilde{\partial}_t$ at each time step. By rewriting q_{t-1} in terms of the previous $\tilde{\partial}_{t-1}$ values, we get the following recurrence:

$$\tilde{\partial}_t(i, \alpha) \iff x_t = \alpha \wedge \bigvee_{\langle j, \beta \rangle \in \delta^{-1}(i)} \tilde{\partial}_t(j, \beta). \quad (33)$$

Since this formula is linearly separable, we can compute it in a single neural network layer from \mathbf{x}_t and \mathbf{h}_{t-1} .

Finally, we consider the base case. We need to ensure that transitions out of the initial state work out correctly at the first time step. We do this by adding a new memory unit f_t to \mathbf{h}_t which is always rewritten to have value 1. Thus, if $f_{t-1} = 0$, we can be sure we are in the initial time step. For each transition out of the initial state, we add $f_{t-1} = 0$ as an additional term to get

$$\tilde{\partial}_t(0, \alpha) \iff x_t = \alpha \wedge (f_{t-1} = 0 \vee \bigvee_{\langle j, \beta \rangle \in \delta^{-1}(0)} \tilde{\partial}_t(j, \beta)). \quad (34)$$

This equation is still linearly separable and guarantees that the initial step will be computed correctly. □

C GRU Lemmas

These results follow similar arguments to those in [Subsection 3.1](#) and [Appendix B](#).

Lemma C.1 (GRU state complexity). *The GRU hidden state has state complexity*

$$\mathfrak{M}(\mathbf{h}_n) = O(1).$$

Proof. The configuration set of \mathbf{z}_t is a subset of $\{0, 1\}^k$. Thus, we have two possibilities for each value of $[\mathbf{h}_t]_i$: either $[\mathbf{h}_{t-1}]_i$ or $[\mathbf{u}_t]_i$. Furthermore, the configuration set of $[\mathbf{u}_t]_i$ is a subset of $\{-1, 1\}$. Let S_t be the configuration set of $[\mathbf{h}_t]_i$. We can describe S_t according to

$$S_0 = \{0\} \quad (35)$$

$$S_t \subseteq S_{t-1} \cup \{-1, 1\}. \quad (36)$$

This implies that, at most, there are only three possible values for each logit: -1 , 0 , or 1 . Thus, the state complexity of \mathbf{h}_n is

$$\mathfrak{M}(\mathbf{h}_n) \leq 3^k = O(1). \quad (37)$$

□

Lemma C.2 (GRU lower bound).

$$\text{RL} \subseteq L(\text{GRU}).$$

Proof. We can simulate a finite-state machine using the $\tilde{\delta}$ construction from [Theorem 3.2](#). We compute values for the following predicate at each time step:

$$\tilde{\delta}_t(i, \alpha) \iff x_t = \alpha \wedge \bigvee_{(j, \beta) \in \delta^{-1}(i)} \tilde{\delta}_{t-1}(j, \beta). \quad (38)$$

Since (38) is linearly separable, we can store $\tilde{\delta}_t$ in our hidden state \mathbf{h}_t and recurrently compute its update. The base case can be handled similarly to (34). A final feedforward layer accepts or rejects according to (32). \square

D Attention Lemmas

Theorem D.1 ([Theorem 4.1](#) restated). *Let t_1, \dots, t_m be the subsequence of time steps that maximize $\mathbf{q}\mathbf{k}_t$. Asymptotically, attention computes*

$$\lim_{N \rightarrow \infty} \text{attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \lim_{N \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m \mathbf{v}_{t_i}.$$

Proof. Observe that, asymptotically, $\text{softmax}(\mathbf{u})$ approaches a function

$$\lim_{N \rightarrow \infty} \text{softmax}(N\mathbf{u})_t = \begin{cases} \frac{1}{m} & \text{if } u_t = \max(\mathbf{u}) \\ 0 & \text{otherwise.} \end{cases} \quad (39)$$

Thus, the output of the attention mechanism reduces to the sum

$$\lim_{N \rightarrow \infty} \sum_{i=1}^m \frac{1}{m} \mathbf{v}_{t_i}. \quad (40)$$

\square

Lemma D.1 ([Theorem 4.2](#) restated). *The full state of the attention layer has state complexity*

$$\mathfrak{M}(\mathbf{V}_n) = 2^{\Theta(n)}.$$

Proof. By the general upper bound on state complexity ([Theorem A.2](#)), we know that $\mathfrak{M}(\mathbf{V}_n) = 2^{O(n)}$. We now show the lower bound.

We pick weights θ in the encoder such that $\mathbf{v}_t = \mathbf{x}_t$. Thus, $\mathfrak{M}(\mathbf{v}_t^\theta) = |\Sigma|$ for all t . Since the values at each time step are independent, we know that

$$\mathfrak{M}(\mathbf{V}_n^\theta) = |\Sigma|^n \quad (41)$$

$$\therefore \mathfrak{M}(\mathbf{V}_n) = 2^{\Omega(n)}. \quad (42)$$

\square

Lemma D.2 ([Theorem 4.3](#) restated). *The attention summary vector has state complexity*

$$\mathfrak{M}(\mathbf{h}_n) = O(n^2).$$

Proof. By [Theorem 4.1](#), we know that

$$\lim_{N \rightarrow \infty} \mathbf{h}_n = \lim_{N \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m \mathbf{v}_{t_i}. \quad (43)$$

By construction, there is a finite set S containing all possible configurations of every \mathbf{v}_t . We bound the number of configurations for each \mathbf{v}_{t_i} by $|S|$ to get

$$\mathfrak{M}(\mathbf{h}_n) \leq \sum_{m=1}^n |S|m \leq |S|n^2 = O(n^2). \quad (44)$$

\square

Lemma D.3 (Attention state complexity lower bound). *The attention summary vector has state complexity*

$$\mathfrak{M}(\mathbf{h}_n) = \Omega(n).$$

Proof. Consider the case where keys and values have dimension 1. Further, let the input strings come from a binary alphabet $\Sigma = \{0, 1\}$. We pick parameters θ in the encoder such that, for all t ,

$$\lim_{N \rightarrow \infty} v_t = \begin{cases} 0 & \text{if } \mathbf{x}_t = \mathbf{0} \\ 1 & \text{otherwise} \end{cases} \quad (45)$$

and $\lim_{N \rightarrow \infty} k_t = 1$. Then, attention returns

$$\lim_{N \rightarrow \infty} \sum_{t=1}^n v_t = \frac{l}{n} \quad (46)$$

where l is the number of t such that $\mathbf{x}_t = \mathbf{1}$. We can vary the input to produce l from 1 to n . Thus, we have

$$\mathfrak{M}(\mathbf{h}_n^\theta) = n \quad (47)$$

$$\therefore \mathfrak{M}(\mathbf{h}_n) = \Omega(n). \quad (48)$$

\square

Lemma D.4 (Attention state complexity with unique maximum). *If, for all \mathbf{X} , there exists a unique t^* such that $t^* = \max_t \mathbf{q}_n \mathbf{k}_t$, then*

$$\mathfrak{M}(\mathbf{h}_n) = O(1).$$

Proof. If $\mathbf{q}_n \mathbf{k}_t$ has a unique maximum, then by [Corollary 4.1.1](#) attention returns

$$\lim_{N \rightarrow \infty} \arg \max_{\mathbf{v}_t} \mathbf{q}_n \mathbf{k}_t = \lim_{N \rightarrow \infty} \mathbf{v}_{t^*}. \quad (49)$$

By construction, there is a finite set S which is a superset of the configuration set of \mathbf{v}_{t^*} . Thus,

$$\mathfrak{P}(\mathbf{h}_n) \leq |S| = O(1). \quad (50)$$

□

Lemma D.5 (Attention state complexity with ReLU activations). *If $\lim_{N \rightarrow \infty} \mathbf{v}_t \in \{0, \infty\}^k$ for $1 \leq t \leq n$, then*

$$\mathfrak{P}(\mathbf{h}_n) = O(1).$$

Proof. By [Theorem 4.1](#), we know that attention computes

$$\lim_{N \rightarrow \infty} \mathbf{h}_n = \lim_{N \rightarrow \infty} \frac{1}{m} \sum_{i=1}^m \mathbf{v}_{t_i}. \quad (51)$$

This sum evaluates to a vector in $\{0, \infty\}^k$, which means that

$$\mathfrak{P}(\mathbf{h}_n) \leq 2^k = O(1). \quad (52)$$

□

[Lemma D.5](#) applies if the sequence $\mathbf{v}_1, \dots, \mathbf{v}_n$ is computed as the output of ReLU. A similar result holds if it is computed as the output of an unsquashed linear transformation.

E CNN Lemmas

Lemma E.1 (CNN counterexample).

$$a^* b a^* \notin L(\text{CNN}).$$

Proof. By contradiction. Assume we can write a network with window size k that accepts any string with exactly one b and reject any other string. Consider a string with two b s at indices i and j where $|i - j| > 2k + 1$. Then, no column in the network receives both \mathbf{x}_i and \mathbf{x}_j as input. When we replace one b with an a , the value of \mathbf{h}_+ remains the same. Since the value of \mathbf{h}_+ (26) fully determines acceptance, the network does not accept this new string. However, the string now contains exactly one b , so we reach a contradiction. □

Definition E.1 (Strictly k -local grammar). A strictly k -local grammar over an alphabet Σ is a set of allowable k -grams S . Each $s \in S$ takes the form

$$s \in (\Sigma \cup \{\#\})^k$$

where $\#$ is a padding symbol for the start and end of sentences.

Definition E.2 (Strictly local acceptance). A strictly k -local grammar S accepts a string σ if, at each index i ,

$$\sigma_i \sigma_{i+1} \dots \sigma_{i+k-1} \in S.$$

Lemma E.2 (Implies [Theorem 5.2](#)). *A k -CNN can asymptotically accept any strictly $2k+1$ -local language.*

Proof. We construct a k -CNN to simulate a strictly $2k+1$ -local grammar. In the convolutional layer (25), each filter identifies whether a particular invalid $2k+1$ -gram is matched. This condition is a conjunction of one-hot terms, so we use tanh to construct a linear transformation that comes out to 1 if a particular invalid sequence is matched, and -1 otherwise.

Next, the pooling layer (26) collapses the filter values at each time step. A pooled filter will be 1 if the invalid sequence it detects was matched somewhere and -1 otherwise.

Finally, we decide acceptance (27) by verifying that no invalid pattern was detected. To do this, we assign each filter a weight of -1 use a threshold of $-K + \frac{1}{2}$ where K is the number of invalid patterns. If any filter has value 1, then this sum will be negative. Otherwise, it will be $\frac{1}{2}$. Thus, asymptotic sigmoid will give us a correct acceptance decision. □

F Neural Stack Lemmas

Refer to [Hao et al. \(2018\)](#) for a definition of the StackNN architecture. The architecture utilizes a differentiable data structure called a *neural stack*. We show that this data structure has $2^{\Theta(n)}$ state complexity.

Lemma F.1 (Neural stack state complexity). *Let $\mathbf{S}_n \in \mathbb{R}^{nk}$ be a neural stack with a feedforward controller. Then,*

$$\mathfrak{P}(\mathbf{S}_n) = 2^{\Theta(n)}.$$

Proof. By the general state complexity bound (Theorem A.2), we know that $\mathfrak{M}(\mathbf{S}_n) = 2^{O(n)}$. We now show the lower bound.

The stack at time step n is a matrix $\mathbf{S}_n \in \mathbb{R}^{nk}$ where the rows correspond to vectors that have been pushed during the previous time steps. We set the weights of the controller θ such that, at each step, we pop with strength 0 and push \mathbf{x}_t with strength 1. Then, we have

$$\mathfrak{M}(\mathbf{S}_n^\theta) = |\Sigma|^n \quad (53)$$

$$\therefore \mathfrak{M}(\mathbf{S}_n) = 2^{\Omega(n)}. \quad (54)$$

□